Ledesma Ayala Angel Yeremi

API REST

Una API REST (Interfaz de Programación de Aplicaciones basada en Transferencia de Estado Representacional) es un conjunto de reglas y convenciones para diseñar y construir servicios web que permiten la comunicación eficiente entre sistemas a través de Internet. Está basada en los principios de la arquitectura REST, propuesta por Roy Fielding en su tesis doctoral.

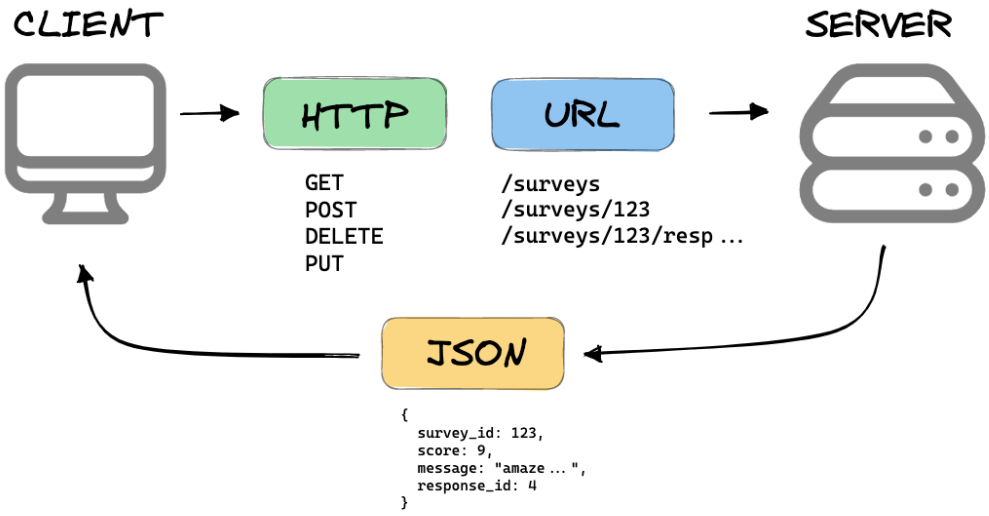
Características clave de una API REST:

1. Transferencia de Estado Representacional (RESTful):
    - Utiliza la transferencia de representaciones de recursos entre el cliente y el servidor. Cada recurso es identificado por una URL única, y las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) se realizan mediante los métodos estándar de HTTP (POST, GET, PUT, DELETE).
2. Recursos:
    - Todo en una API REST es considerado como un recurso, que puede ser cualquier entidad o concepto significativo en el contexto de la aplicación. Por ejemplo, un recurso puede representar un usuario, un producto, una publicación, etc.
3. Independencia de Estado:
    - Cada solicitud del cliente al servidor contiene toda la información necesaria para comprender y procesar la solicitud. La independencia de estado significa que cada solicitud es autónoma y no depende del estado anterior.
4. Protocolo HTTP/HTTPS:
    - Utiliza el protocolo HTTP (o su versión segura, HTTPS) para la comunicación entre el cliente y el servidor. Cada recurso tiene una URL única y las operaciones CRUD se realizan mediante los métodos HTTP correspondientes.
5. Formatos de Representación:
    - Los datos intercambiados entre el cliente y el servidor se presentan comúnmente en formatos ligeros y legibles, como JSON (JavaScript Object Notation) o XML (eXtensible Markup Language).
6. Sin Estado (Stateless):

- Cada solicitud desde un cliente al servidor contiene toda la información necesaria. El servidor no guarda información sobre el estado del cliente entre las solicitudes.

7. Lenguaje Agnóstico:
    - REST es agnóstico respecto al lenguaje de programación y puede ser implementado en cualquier plataforma que admita la comunicación a través de HTTP.

Las API REST son utilizadas para facilitar la integración y la interacción entre sistemas heterogéneos, como aplicaciones web, móviles o servicios en la nube. Proporcionan una arquitectura flexible y escalable que se ha vuelto ampliamente adoptada en el desarrollo de servicios web.



Ledesma Ayala Angel Yeremi

Principios RESTful:

Los principios RESTful son los fundamentos sobre los cuales se construye la arquitectura REST. Estos principios proporcionan pautas para diseñar sistemas web eficientes y escalables:

1. Transferencia de Estado Representacional (REST):
    - Los sistemas RESTful utilizan la transferencia de representaciones de recursos entre el cliente y el servidor. Cada recurso es único y se identifica mediante una URL.
2. Cliente-Servidor:
    - La arquitectura se divide en un componente cliente y un componente servidor, cada uno de los cuales es independiente y puede evolucionar por separado sin afectar al otro. La comunicación entre ellos se realiza mediante solicitudes y respuestas.
3. Sin Estado (Stateless):
    - Cada solicitud del cliente al servidor contiene toda la información necesaria para entender y procesar la solicitud. El servidor no guarda información sobre el estado del cliente entre las solicitudes.
4. Interfaz Uniforme:
    - La interfaz entre el cliente y el servidor debe ser uniforme y simplificada. Esto incluye la identificación de recursos a través de URIs, la manipulación de recursos mediante representaciones y la navegación entre recursos a través de hipermedios (HATEOAS).
5. Capacidad de Cache:
    - Las respuestas de las solicitudes deben ser explícitas sobre si se pueden almacenar en caché o no. Esto mejora la eficiencia y reduce la carga en el servidor.
6. Sistema de Capas:
    - La arquitectura puede estar compuesta por varias capas (por ejemplo, servidores intermedios, gateways, etc.), y cada capa realiza una función específica. Esto proporciona modularidad y escalabilidad.

Ledesma Ayala Angel Yeremi

Concepto de recursos en REST:

En el contexto de REST, un recurso es cualquier entidad o concepto que tenga sentido en el dominio de la aplicación y que pueda ser identificado de manera única. Los recursos representan información que puede ser manipulada a través de la API. Algunos ejemplos de recursos pueden ser usuarios, productos, publicaciones o cualquier otra entidad relevante para la aplicación.

La idea central es que, en lugar de centrarse en las acciones que se pueden realizar (como en las arquitecturas RPC - Remote Procedure Call), REST se centra en los datos y recursos, permitiendo que las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) se realicen sobre esos recursos.

Diseño de URIs significativas:

La identificación única de los recursos se logra mediante Uniform Resource Identifiers (URIs), que son direcciones web que apuntan a un recurso específico. Al diseñar URIs en una API REST, es importante seguir prácticas que faciliten la comprensión y el mantenimiento:

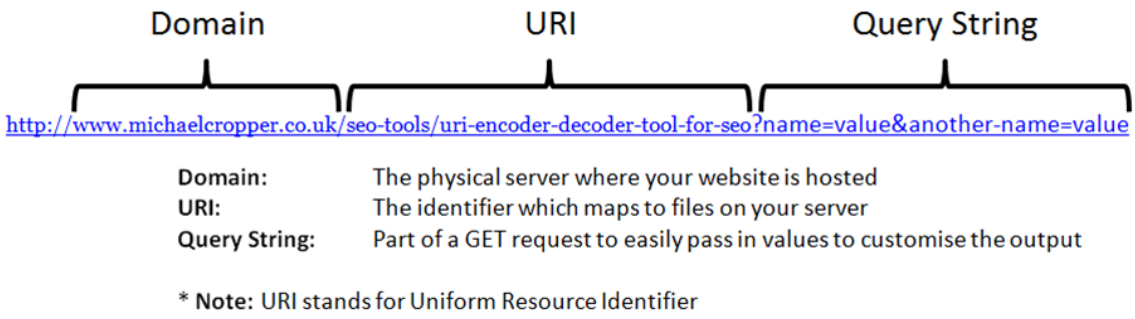
1. Jerarquía de Recursos:
    - Organizar los recursos de manera jerárquica en la URI puede reflejar la relación entre ellos. Por ejemplo, /usuarios podría ser la colección de usuarios, y /usuarios/123 sería un usuario específico con el ID 123.
2. Nombres Plurales:
    - Utilizar nombres en plural para las colecciones de recursos (por ejemplo, /usuarios) para indicar que se trata de una colección de múltiples elementos.
3. Evitar Acciones en las URIs:
    - Evitar incluir acciones o verbos en las URIs. En lugar de /obtener-usuario/123, es preferible /usuarios/123 para obtener un usuario específico.
4. Utilizar Guiones Bajos o Guiones en Caso de Nombres Compuestos:
    - Al representar espacios o nombres compuestos, se pueden usar guiones bajos o guiones. Por ejemplo, /publicaciones/123 en lugar de /posts/123.
5. Evitar Uso de Números de Identificación en URIs:
    - Utilizar identificadores significativos o nombres en lugar de números en las URIs siempre que sea posible. Por ejemplo, /usuarios/alice en lugar de /usuarios/123.

6. Versionamiento en URIs:
    - Considerar estrategias de versionamiento en las URIs, como /v1/usuarios/123, para manejar cambios en la API sin afectar las versiones existentes.

Un ejemplo de diseño de URI significativa podría ser:

- /usuarios: Colección de usuarios.
- /usuarios/123: Usuario con ID 123.
- /productos: Colección de productos.
- /productos/abc123: Producto con ID abc123.

El diseño cuidadoso de URIs facilita la comprensión de la estructura de la API y mejora la experiencia del desarrollador al interactuar con los recursos.



Métodos HTTP en REST:

En la arquitectura REST, los métodos HTTP (Hypertext Transfer Protocol) juegan un papel crucial para definir las operaciones que se realizarán sobre los recursos. Los principales métodos utilizados en REST están asociados a las operaciones CRUD (Crear, Leer, Actualizar, Eliminar):

1. GET (Leer):
    - El método GET se utiliza para recuperar información de un recurso. Cuando se realiza una solicitud GET a una URI específica, se espera que el servidor devuelva la representación del recurso solicitado. No debe tener efectos secundarios en el servidor.
2. POST (Crear):
    - El método POST se utiliza para enviar datos al servidor para crear un nuevo recurso. La URI a la que se envía la solicitud POST generalmente apunta a la colección donde se creará el recurso.
3. PUT (Actualizar):

- El método PUT se utiliza para actualizar o crear un recurso en una URI específica. La solicitud PUT debe contener la representación completa del recurso, y el servidor la utilizará para actualizar el recurso existente o crear uno nuevo si no existe.

4. DELETE (Eliminar):

    - El método DELETE se utiliza para solicitar la eliminación de un recurso en una URI específica. Después de una solicitud DELETE exitosa, el recurso debe ser eliminado o marcado como eliminado en el servidor.

Estos métodos reflejan las operaciones fundamentales de CRUD sobre los recursos. Es importante notar que el diseño de una API RESTful implica asignar estas operaciones a los recursos de manera coherente y significativa.

Representaciones y Formatos de Datos:

En el contexto de las API REST, la representación de datos se refiere a cómo se estructuran y presentan los datos que se envían entre el cliente y el servidor. Dos formatos comunes para la representación de datos son JSON (JavaScript Object Notation) y XML (eXtensible Markup Language).

Diferentes formatos de representación de datos (JSON, XML):

1. JSON (JavaScript Object Notation):
    - JSON es un formato ligero y fácil de leer que utiliza una estructura de pares clave-valor. Es ampliamente utilizado en el desarrollo web y es fácilmente interpretable por humanos y máquinas.

    Ejemplo de un objeto JSON representando un usuario:

```
{    "id": 123,
     "nombre": "Ejemplo Usuario",
     "correo": usuario@example.com
}
```

2. XML (eXtensible Markup Language):
    - XML utiliza una estructura de marcas etiquetadas para definir documentos. Aunque es más verboso en comparación con JSON, XML es legible por máquinas y es ampliamente utilizado en diversos contextos.

    Ejemplo de un elemento XML representando un usuario:

```
<usuario>
        <id>123</id>
    <nombre>Ejemplo Usuario</nombre>
        <correo>usuario@example.com</correo>
</usuario>
```

Selección del formato adecuado según el contexto:

La elección entre JSON y XML depende del contexto de la aplicación y las preferencias del desarrollador. Algunos factores a considerar incluyen:

1. Legibilidad:
    - JSON tiende a ser más conciso y legible para los humanos debido a su sintaxis más simple.
2. Verbosidad:
    - XML es más verboso que JSON, lo que significa que puede haber más texto para representar la misma información.
3. Uso en la Web:

- JSON es más comúnmente utilizado en aplicaciones web modernas, especialmente en el desarrollo de aplicaciones basadas en JavaScript.

4. Soporte y Herramientas:
    - Ambos formatos son compatibles con una amplia gama de lenguajes de programación y tienen herramientas disponibles para procesarlos.
5. Estructura de Datos:
    - La estructura de datos y los requisitos de la aplicación pueden influir en la elección del formato. Algunas aplicaciones prefieren la estructura jerárquica de XML, mientras que otras encuentran más conveniente la simplicidad de JSON.

En general, JSON se ha vuelto más popular en el desarrollo de API REST debido a su simplicidad, legibilidad y la prevalencia de JavaScript en el entorno web. Sin embargo, la elección entre JSON y XML depende de las necesidades específicas del proyecto y las preferencias del equipo de desarrollo.

Estructura de Solicitudes y Respuestas:

La estructura de las solicitudes y respuestas en una API REST es fundamental para la comunicación eficiente entre el cliente y el servidor. Ambos aspectos, las solicitudes y las respuestas, siguen un formato específico para garantizar una comprensión clara y una manipulación efectiva de los recursos.

Anatomía de una solicitud HTTP RESTful:

1. Método HTTP:
    - Indica la operación que se realizará sobre el recurso. Puede ser GET, POST, PUT, DELETE, u otros métodos estándar de HTTP.
2. Encabezados (Headers):
    - Proporcionan información adicional sobre la solicitud, como el tipo de contenido aceptado o el formato de respuesta deseado.
3. Cuerpo de la Solicitud (Request Body):
    - Contiene los datos que se envían al servidor en una solicitud POST o PUT. En una solicitud GET, este campo generalmente está ausente.

Estructura típica de respuestas, incluyendo códigos de estado HTTP:

1. Código de Estado HTTP:
    - Indica el resultado de la solicitud. Los códigos de estado más comunes incluyen 200 (OK), 201 (Creado), 204 (Sin Contenido), 400 (Solicitud

Incorrecta), 404 (No Encontrado), 500 (Error Interno del Servidor), entre otros.

2. Encabezados de Respuesta (Response Headers):
    - Proporcionan información sobre la respuesta, como el tipo de contenido devuelto o la longitud del contenido.
3. Cuerpo de la Respuesta (Response Body):
    - Contiene los datos devueltos por el servidor. Puede estar en formato JSON, XML u otro formato según la negociación de contenido.

Autenticación y Autorización:

La autenticación y la autorización son aspectos cruciales en el diseño y la implementación de una API REST para garantizar la seguridad y la protección de los recursos. Estos dos conceptos son distintos pero están estrechamente relacionados en la gestión del acceso a los servicios y datos.

Métodos comunes de autenticación en REST:

1. Token de Acceso (Access Token):
    - Un token de acceso es una cadena de caracteres que se utiliza para autenticar a un usuario. Este token se obtiene generalmente después de un proceso de inicio de sesión exitoso y se incluye en las solicitudes al servidor para demostrar que el usuario está autorizado.

Ejemplo de solicitud con token de acceso:

GET /recursos/123 Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

2. Clave de API (API Key):
    - Una clave de API es una cadena única que se proporciona a los desarrolladores para identificar su aplicación y autenticar las solicitudes. La clave se envía generalmente como parte de la solicitud, a menudo como un encabezado.

GET /recursos/123 X-Api-Key: abcdef123456

3. Usuario y Contraseña (Basic Authentication):
    - En este método, el cliente envía un nombre de usuario y una contraseña codificados en Base64 en la solicitud. Aunque es simple, se debe usar junto con una capa de seguridad adicional, como HTTPS, para proteger las credenciales.

GET /recursos/123 Authorization: Basic dXNlcm5hbWU6cGFzc3dvcmQ=

Estrategias para autorización y control de acceso:

1. Control de Acceso basado en Roles (Role-Based Access Control - RBAC):
    - Asigna roles a los usuarios y luego define permisos para cada rol. Los usuarios obtienen acceso según su rol.
2. Control de Acceso basado en Token (Token-Based Access Control):
    - Utiliza tokens con información de autorización específica para controlar el acceso a recursos. Los tokens pueden contener roles, permisos u otra información relevante.
3. Políticas de Autorización:
    - Define políticas específicas que determinan quién tiene acceso a qué recursos. Puede basarse en criterios como el rol del usuario, la propiedad del recurso o condiciones personalizadas.
4. OAuth (Open Authorization):
    - Un protocolo de autorización que permite que una aplicación de terceros obtenga acceso limitado a los recursos en nombre de un usuario, sin revelar las credenciales del usuario.

Paginación y Ordenamiento:

En el contexto de API REST, la paginación y el ordenamiento son estrategias esenciales para manejar grandes conjuntos de datos de manera eficiente y brindar a los clientes la capacidad de recuperar la información de manera controlada.

Estrategias para manejar grandes conjuntos de datos:

1. Paginación:
    - La paginación divide grandes conjuntos de datos en fragmentos más pequeños, o "páginas", que se pueden solicitar y procesar de manera independiente. Esto es particularmente útil cuando se trata con grandes cantidades de recursos para evitar la carga excesiva en una sola solicitud.

      GET /recursos?pagina=2&cantidad=10

    - En este ejemplo, se solicitan 10 recursos, comenzando desde la segunda página.
2. Ordenamiento:
    - El ordenamiento permite que los resultados de una solicitud se presenten en un orden específico, generalmente ascendente o descendente según un atributo determinado. Esto es útil cuando los recursos tienen un atributo significativo para el ordenamiento.

GET /recursos?orden=nombre

- En este ejemplo, se solicitan los recursos ordenados alfabéticamente por el atributo "nombre".

3. Cursor de Paginación:

   - En lugar de utilizar números de página, algunas API utilizan un cursor de paginación, que es una referencia a un recurso específico en el conjunto de datos. Esto puede proporcionar una forma más eficiente de navegar a través de grandes conjuntos de datos, especialmente cuando los recursos no son estáticos.

   GET /recursos?cursor=abc123

   - En este ejemplo, se solicitan los recursos que siguen al recurso identificado por el cursor "abc123".

4. Filtros:

   - Además de paginación y ordenamiento, los filtros permiten a los clientes especificar criterios específicos para limitar los resultados a recursos que cumplan con ciertas condiciones.

   GET /recursos?categoria=tecnologia

   - En este ejemplo, se solicitan los recursos que pertenecen a la categoría "tecnología".